Syntelos

SAuth 1.0

Tuesday, 3 February 2009

Overview

SAuth is an HTTP request signing scheme for User Agent authentication. It applies RSA, SHA-1 and MD5 to HTTP request signing.

The User Agent is authenticated and the content of the HTTP request are protected from tampering (middle man).

It has been designed for a system with shared RSA keys on client and server, substantial key sizes (e.g., 1024), and for the very frequent use of those Key Pairs. It compromises cost and benefit in favor of long term security. (RSA signing is expensive).

Other request signing authentication schemes include for example the Amazon S3 authentication scheme. One of the design objectives in request signing schemes is the avoidance of the overhead of challenge- response protocols. A request is statelessly verified with a shared key in a single transaction.

SAuth is not asymmetric in the verification of request signatures. It transmits a reduced form of the signature that is unable to support asymmetric signature verification. The reduced form of the signature enhances the security of the Key Pair at the expense of the cost of verification.

Synopsis

A shared RSA key pair is employed to sign the SHA-1 hash of components of the HTTP Request. The signature of the hash of the request is reduced to 64 bits with a XOR "barrel function" or "shift register".

A header named "SAuth" identifies the authentication scheme. A header named "SAuth-UID" identifies the User Agent with respect to the target host. A header named "SAuth-Nonce" serves to protect the request from replay, and protects the key as well. A header named "SAuth-Signature" includes the authentication signature for this request.

A complete request appears as follows.

```
GET /s/system.pfx
Host: www.syntelos.com
Date: Tue, 27 Jan 2009 03:02:12 GMT
Authorization: SAuth
SAuth: 1.0 RSA SHA-1
SAuth-UID: system
SAuth-Nonce: 83295bf2d7286cd5
SAuth-Signature: 3e96cd0e86d2fa70
```

A client should employ a header named "Authorization" with the value "SAuth". This is standard HTTP, but not required for authentication via SAuth.

When the Request includes an Entity Body, the content is described with one or more of the standard HTTP headers as follows, and is hashed with MD5.

```
Content-Type
Content-Length
Content-Encoding
Content-Range
Content-Location
ETag
Last-Modified
Expires
Content-MD5
```

The SAuth process employs these content headers when the value of the Content Length is recognized as a non zero, positive integer.

A complete example appears immediately following.

```
PUT /test.txt
Host: www.syntelos.com
Date: Tue, 27 Jan 2009 03:14:25 GMT
Content-Type: text/plain
Content-Length: 14
Content-MD5: 4d3b291c116aa0508c2833ad052f4c94
Authorization: SAuth
SAuth: 1.0 RSA SHA-1
SAuth-UID: system
SAuth-Nonce: 6b6b79d4a9432c16
SAuth-Signature: ca63fff1f952e681

hello, world.
```

Signing

In the basic or common form of an authenticated request (without a Request Entity Body), the elements that are included in the hash are (in hash order)

the Request Line Method and Path (not including Protocol),
a header named "Host",
a header named "Date",
a header named "SAuth",
a header named "SAuth-UID",
and a header named "SAuth-Nonce".

These headers are required in the request (in any order).

The Date value must be in small variance from the server side date. The server's accepted variance is intended as a configurable variable that can be tuned, however the idea is some small number of seconds less than five or ten.

Header 'SAuth'

The value of this header is strictly "1.0 RSA SHA-1". Algorithm alias alternatives are not supported.

Header 'SAuth-UID'

The value of this header is the UID string employed when the User Agent principal and key set were created.

The UID is stored in the PKCS#12 PFX set as the "friendly name", and in the X.509 Public Key Certificate as the ASN.1 UID.

Header 'SAuth-Nonce'

The value of this header is relatively random material, encoded in hexadecimal, and must have at least fifteen significant bits. This randomness is relative to the Key Pair and Request Date.

This header permits key protection by adding relatively random information into the hash. The client should not use the same value more than once.

The remainder of the elements of the request for hashing are standard to HTTP. They are hashed as "full line" strings (excepting Request Line Protocol) in order as in the following.

```
GET /s/system.pfx
Host: www.syntelos.com
Date: Tue, 27 Jan 2009 03:02:12 GMT
SAuth: 1.0 RSA SHA-1
SAuth-UID: system
SAuth-Nonce: 83295bf2d7286cd5
```

The corresponding proceedure is represented in the following "illustrated" pseudo code.

```
static void SAuth(HttpRequest request, KeyPair keypair){

    SHA1 sha = new SHA1();
    sha.update("GET /s/system.pfx");
    sha.update("Host: www.syntelos.com");
    sha.update("Date: Tue, 27 Jan 2009 03:02:12 GMT");
    sha.update("SAuth: 1.0 RSA SHA-1");
    sha.update("SAuth-UID: system");
    sha.update("SAuth-Nonce: 83295bf2d7286cd5");

    RSA rsa = new RSA(keypair);
    byte[] signature = rsa.sign(sha);
    long folded = XorFold64(signature);
    String sAuthSignature = hex.encode(folded);

    request.setHeader("SAuth-Signature",sAuthSignature);
}
```

XorFold64

This inexpensive function reduces the signature to 64 bits. It further protects the Private Key under frequent use.

The following Java code represents this function.

```
static long XorFold64 ( byte[] b){
    if ( null == b)
        return 0L;
    else {
        long accum = 0, tmp;
        long ch;
        int shift ;
        int c = 0, uz = b.length, uc = (uz- 1);
        for ( ; c < uz; c++, uc--)
        {
            ch = b[c] & 0xff;
            shift = ((uc % 8) * 8);
            tmp = (ch << shift);
            accum ^= tmp;
        }
        return accum;
    }
}
```

Where 'b' is the (numeric or unencoded) hash digest value.

The shift register folds its input onto itself. Where its input is greater than eight bytes, additional bytes are XOR'ed with their positional ancestors. As the signature produced with a 1024- bit RSA key is also 1024 bits, the reduction / protection effect of the shift register is significant (over 64 bits).

As shown above, the 64 bit output of the Xor Fold function is hex encoded.

Authenticating

The Server verifies the Signature by repeating the SAuth signing proceedure with its copy of the identified Key Pair.

The Server must validate the variance in the Date value, and the bit length of the Nonce value.

The Server uses a numeric comparison on the User Agent's and Server's Signature values to know that the client has a copy of the Key Pair identified by the (Host and) UID values. The User Agent is authenticated with an exact match in this comparison.

Request Entity Body

When a header named "Content-Length" is present and has a value recognized as a positive integer, the Request is recognized to have a (non empty) Entity Body.

In this case, the following proceedure is required to authenticate with SAuth.

The following HTTP standard headers are included in the Request Hash when present, in the following order. They are included in the Request Hash after the "Date" and before "SAuth".

```
Content-Type
Content-Length
Content-Encoding
Content-Range
Content-Location
ETag
Last-Modified
Expires
Content-MD5
```

The required header named "Content-Length" is included in the hash after "Date", after any (optionally) present "Content-Type", and before the required header "Content-MD5".

A header named "Content-MD5" is required. It is included in the SAuth hash after "Content-Length", and before "SAuth".

Header 'Content-MD5'

The value of this header is the hexadecimal- encoded product of the MD5 hash of the Request Entity Body.

The corresponding proceedure is represented in the following "illustrated" pseudo code.

```
static void SAuth(HttpRequest request, KeyPair keypair){

    MD5 md = new MD5();
    md.update(request.getBody());
    String mdHashHex = md.digestHex();

    request.setHeader("Content-MD5",mdHashHex);

    SHA1 sha = new SHA1();
    sha.update("PUT /test.txt");
    sha.update("Host: www.syntelos.com");
    sha.update("Date: Tue, 27 Jan 2009 03:14:25 GMT");
    sha.update("Content-Type: text/plain");
    sha.update("Content-Length: 14");
    sha.update("Content-MD5: "+mdHashHex);
    sha.update("SAuth: 1.0 RSA SHA-1");
    sha.update("SAuth-UID: system");
    sha.update("SAuth-Nonce: 6b6b79d4a9432c16");

    RSA rsa = new RSA(keypair);
    byte[] signature = rsa.sign(sha);
    long folded = XorFold64(signature);
    String sAuthSignature = hex.encode(folded);

    request.setHeader("SAuth-Signature",sAuthSignature);
}
```

Response Codes

The Server accepts authentication with a response appropriate to the protocol of the operation.

Unauthorized

The Server must deny a request for authentication with the HTTP Response Status Code value of 401, "Unauthorized". As HTTP requires the 401 to include a header named "WWW-Authenticate", this header must indicate the challenge clearly using the following information.

```
HTTP/1.1 401 Unauthorized
Date: Tue, 27 Jan 2009 01:12:23 GMT
WWW-Authenticate: SAuth realm="syntelos.com",uid="system"
```

If the UID is unknown, the token pair shall be omitted as in the following example.

```
HTTP/1.1 401 Unauthorized
Date: Tue, 27 Jan 2009 01:12:23 GMT
WWW-Authenticate: SAuth realm="syntelos.com"
```

Bad Request

Incorrect use of the SAuth headers, including a missing or malformed header, must produce a Response with a Status Code value of 400, "Bad Request".

Implementation Notes

HTTP extension protocols may include additional elements in the request hash as required to secure the integrity of the request.

Review

Design

The security needs for design of SAuth include the protection of the key pair, the integrity of the request, and authentication.

In the system for which this scheme has been designed, the User Agent Key Pair is employed with very high frequency and for many purposes. Its protection is fairly critical. While the use of the Xor Fold function precludes an asymmetric signature verification (implementation of SAuth), a highly folded signature adds a significant layer to the protection of the key pair.

Each of these three design needs are believed to be very well, if expensively, met with SAuth. It is heavy and strong for applications that require this kind of authentication, for example server software upgrades, restarts and shutdowns.

Request signing

HTTP request signing schemes endeavor to avoid the overhead of challenge- response protocols. The objective in using or creating a request signing scheme is the server side authentication of the client in one request.

While request signing wants to be especially fast to verify on the server side, this scheme has compromised cost for benefit in favor of the long term protection of the User Agent key pair and system security.

Alternatives

The methods defined for standard HTTP Authentication, in particular "Digest", are challenge response schemes that would impose undesireable TCP overhead. Request signing schemes conserve connections and threads.

Other schemes in RFC and Draft form (including SSL/TLS) are either too complex, too imposing with requirements, or more expensive than SAuth.

The one practical request signing scheme that could have been used in place of this one, the Amazon S3 scheme protects the Secret Key less well than SAuth.

Weaknesses

The Date variance window is the single most critical source of security issues. SAuth provides the tools needed for the Server to implement a variety of reinforcing schemes.

The expense of RSA signing is significant. Nonetheless, RSA signing has been retained in the scheme for the benefit of long term security.

Server side design intent

The server creates a User Agent with one key pair that has many applications throughout the system, including both communication and content protection.

The server delivers a created Key Pair to the client in a PKCS#12 (PFX) package, and maintains the key pair independently under its protection. Shared keys permit stateless methods of operation compatible with RESTful HTTP, as well as additional system and application features.

Client side design intent

The scheme is compatible with implementation in JavaScript (likewise ActionScript), the most challenging case known to this writer.

Author's address

John Pritchard
Syntelos
jdp@syntelos.com

References

HTTP - RFC 2616
HTTP Authentication - RFC 2617
AWS S3 Authentication - Amazon S3 Developer Guide
RSA, SHA, and MD5 - Applied Cryptography, Rivest, Second Edition